Teoría de Lenguajes - Segundo Parcial

Primer cuatrimestre de 2020

Apagar los celulares.
Hacer cada ejercicio en hojas separadas.
Poner nombre, número de orden y número de página en cada ejercicio.
Justificar todas las respuestas.
El examen es a libro abierto.
Se aprueba con 65 ó más puntos.

1. (34 pts) Dada la gramática: $G = \langle \{S, A, B, C, D\}, \{2, 3\}, S, P \rangle$, con P:

$$
\begin{array}{l}
S \to 3 \mid A \mid 3A \\
A \to 2C \mid C \\
B \to \lambda \mid B2 \mid SS \\
C \to 2 \mid S \\
D \to DB \mid A3
\end{array}
$$

   a) Aplicar la eliminación de símbolos de modo que queden eliminadas producciones innecesarias sin modificar ni agregar otras, y de modo que se preserve el lenguaje $L(G)$.

   b) Determinar si la gramática resultante de (a) es SLR, indicando todos los conflictos que pudiera haber asociados a la tabla.

   c) Determinar si la gramática resultante de (a) es LR(0).

2. (33 pts) Considerar un lenguaje de programación reducido que usa sólo el tipo entero, en el que un programa es un secuencias de 0 o más sentencias separadas estas por *;*, donde cada sentencia puede ser:

   - asignación (múltiple) de un número a una o más variables, de la forma *var = var = ... = var = num*
   - asignación (múltiple) de una variable a otras, de la forma *var = var = ... = var*
   - incremento del valor de una variable, de la forma *var++*
   - impresión del valor de una variable, de la forma *pr var*

   El token *num* tiene un atributo numérico que representa un valor entero. Asumir que las variables arrancan con valor 0.

   Se pide definir una GLC sin símbolos inútiles, utilizando los atributos que se considere necesarios, de modo que quede definida una TDS que permita que las asignaciones e impresiones funcionen del modo usual, y que además al final de la ejecución de la secuencia se imprima el máximo valor que haya sido impreso anteriormente (si no se imprimió nada antes, se imprimirá un valor arbitrario o nada). Como código dentro de la TDS se pueden utilizar sentencias de C habituales y diccionarios accesibles globalmente (no otras funciones externas ni bibliotecas).

   Ejemplo: *b = c = 5; a = 3; b = a; b++; pr c; pr b* es una cadena válida, que al ejecutarse imprime *5 4 5*.

3. (33 pts) Sea G la gramática extendida $\langle\{S, B\}, \{a, b, c, `, ', d\}, S, P\rangle$, con P:

$$S \to a(c?aB \mid aB)^* d(c?)^+ c$$
$$B \to (b,)^* b$$

   a) Determinar si G es ELL(1). Si no lo es, dar otra gramática G' que sí lo sea y tal que $L(G) = L(G')$.

   b) Dar el parser iterativo-recursivo para G si G fuera ELL(1) o para G' en otro caso.

Comentarios ejercicio 1:
Todo ok

Ejercicio 1-

a) Para eliminar inútiles, primero elimino inactivos y luego inalcanzables.

- Calculo los símbolos activos:

Sea N el conjunto de activos, arranco con $N = \emptyset$

Como $(S \to 3) \in P$ y $3 \in (N \cup V_T)^*$, $N = \{S\}$

Como $(B \to \lambda) \in P$ y $\lambda \in (N \cup V_T)^*$, $N = \{S, B\}$

Como $(C \to 2) \in P$ y $2 \in (N \cup V_T)^*$, $N = \{S, B, C\}$

Como $(A \to 2C) \in P$ y $2C \in (N \cup V_T)^*$, $N = \{S, A, B, C\}$

Como $(D \to A3) \in P$ y $A3 \in (N \cup V_T)^*$, $N = \{S, A, B, C, D\}$

Como $N = V_N$, todos los símbolos son activos

Ahora calculo los alcanzables

Sea N el conjunto de alcanzables, arranco con $N = \{S\}$

Como $(S \to A) \in P$ y $S \in N$, $N = \{S, A\}$

Como $(A \to C) \in P$ y $A \in N$, $N = \{S, A, C\}$

- Como no hay más producciones que agreguen símbolos a N, llegamos a que S, A y C son alcanzables pero B y D son inalcanzables.

Eliminando B y D, la gramática queda $G' = \langle \{S, A, C\}, \{2, 3\}, S, P' \rangle$ con P':

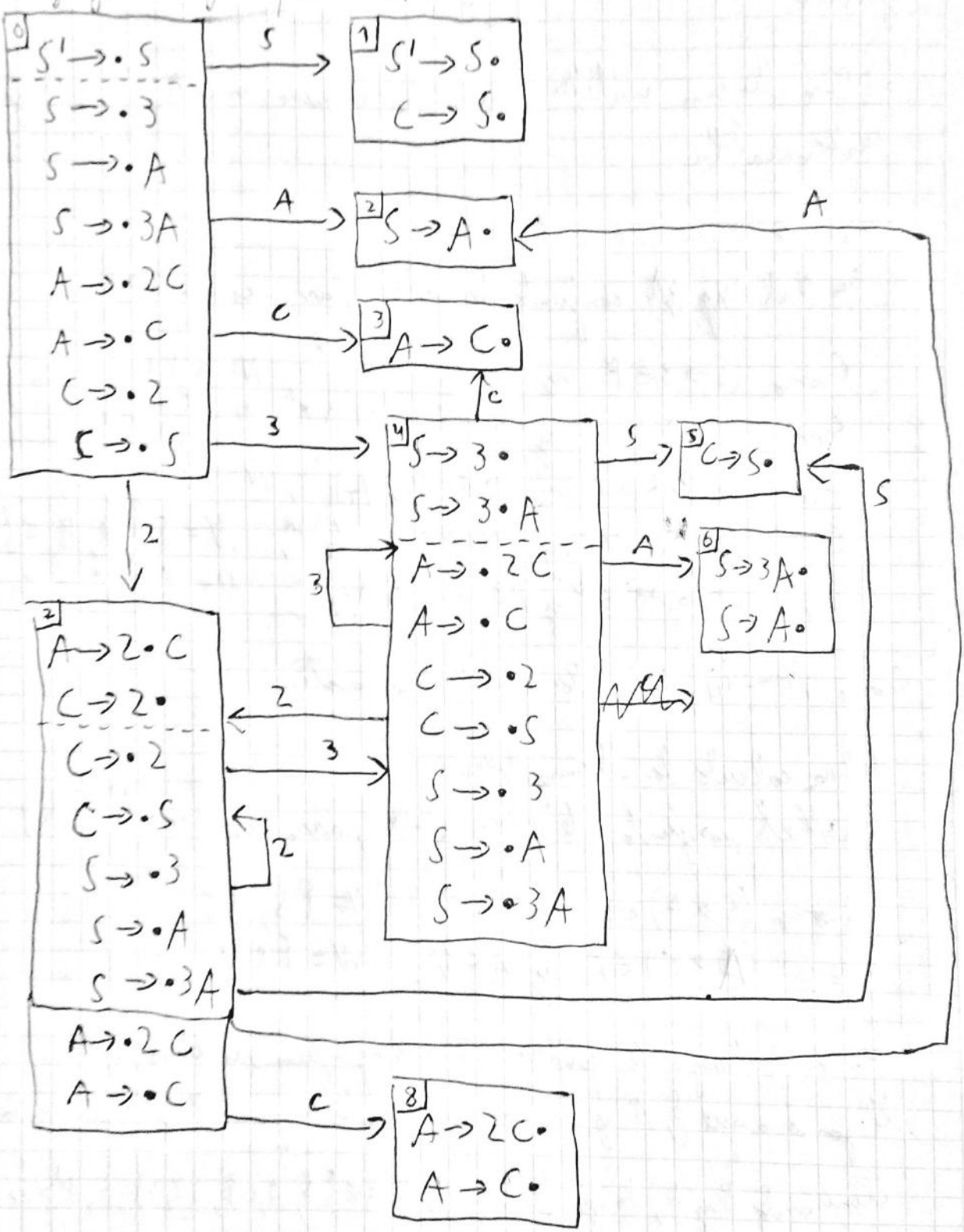
$S \to 3 \mid A \mid 3A$

$A \to 2C \mid C$

$C \to 2 \mid S$

b) Haré el AFD necesario para las tablas LR(0)/SLR:

Agrego S' y la producción S' → S.



Para hacer las tablas SLR, calculo Siguientes

Arranco con Siguientes(S) = {$} y Siguientes(A) = Siguientes(C) = {}

Pq $(S \to A) \in P'$, agrego Sig(S) a Sig(A) $\rightsquigarrow$ Sig(A) = {$}

Pq $(A \to 2C) \in P'$, agrego Sig(A) a Sig(C) $\rightsquigarrow$ Sig(C) = {$}

Las producciones $S \to 3A$, $A \to C$, $C \to 5$ no modifican Siguientes.
$S \to A$ y $A \to 2C$ ahora tampoco. Luego, Sig(S) = Sig(A) = Sig(C) = {$}.

Hago las tablas SLR

Estado	acción 2	3	$	ir_a S	A	C
0	desplazar 7	desplazar 4		1	2	3
1			aceptar reducir $C \to S$			
2			reducir $S \to A$			
3			reducir $A \to C$			
4	desplazar 7	desplazar 4	reducir $S \to 3$	5	6	3
5			reducir $C \to S$			
6			reducir $S \to 3A$ reducir $S \to A$			
7	desplazar 7	desplazar 4	reducir $C \to 2$	5	2	8
8			reducir $A \to 2C$ reducir $A \to C$			

La tabla acción tiene 3 conflictos en acción[1][$], acción[6][$] y acción[8][$]. Luego, G' no es SLR

c) Como LR(0) $\subset$ SLR y G' $\notin$ SLR, G' tampoco es LR(0).

Comentarios ejercicio 2:
Todo ok

Ejercicio 2-

Defino la gramática TDS $G = <\{Prog, S', S, A, A', I, Pn\}$,
$\{var, num, ;, =, ps, +\}$
$Prog$,
$P>$

con atributos

$S'.hubo Pn$	bool	heredado
$S'.max Pn$	int	heredado
$S.hubo Pn$	bool	sintetizado
$S.max Pn$	int	sintetizado
$Pn.valor$	int	sintetizado
$A'.valor$	int	sintetizado

y P producciones

$Prog \rightarrow$ { global.variables = dicc;
$S'.hubo Pn = false;$
$S'.max Pn = 0;$ } S'

$S' \rightarrow \lambda$ { if ($S'.hubo Pn$) print ($S'.max Pn$); }

| $S;$ { $S'_2.hubo Pn = S'.hubo Pn \,||\, S.hubo Pn;$
if ($S'.hubo Pn$ && $S.hubo Pn$) {
$S'_2.max Pn = max(S.max Pn, S'.max Pn)$;
} else if ($S'.hubo Pn$) {
$S'_2.max Pn = S'.hubo Pn;$
} else {
$S'_2.max Pn = S.max Pn$
}
} S'_2

$S \rightarrow A \quad \{ S.huboPn = false;\ S.maxPn = 0 \}$

$\mid \quad I \quad \{ S.huboPn = false;\ S.maxPn = 0 \}$

$\mid \quad Pn \quad \{ S.huboPn = true;\ S.maxPn = Pn.valor \}$

$A \rightarrow var = A' \quad \{ global.variables[var.nombre] = A'.valor \}$

$A' \rightarrow var = A'_2 \quad \{ global.variables[var.nombre] = A'_2.valor;$
$A'.valor = A'_2.valor \}$

$\mid var \quad \{ A'.valor = global.variables[var.nombre] \}$

$\mid num \quad \{ A'.valor = num.valor \}$

$I \rightarrow var++ \quad \{ global.variables[var.nombre]++ \}$

↳ asumo que si no existía, inicia en 0 como hace el map de la STL de C++.

$Pn \rightarrow pr\ var \quad \{ print(global.variables[var.nombre]);$
$Pn.valor = global.variables[var.nombre]; \}$

Comentarios ejercicio 3:
Muy bien la parte a

Antes del accept() deberias matchear "$", sino podes aceptar cadenas del tipo AB donde A pertence a L pero AB no

Ejercicio 3-

a) Comienzo des-extendiendo G:

$S \to a\ A\ d\ C\ c$
$A \to A'\ A \mid \lambda$
$A' \to C'\ a\ B \mid a\ B$
$C \to C'\ C''$
$C' \to c \mid \lambda$
$C'' \to C'\ C'' \mid \lambda$
$B \to B'\ b$
$B' \to b,\ B' \mid \lambda$

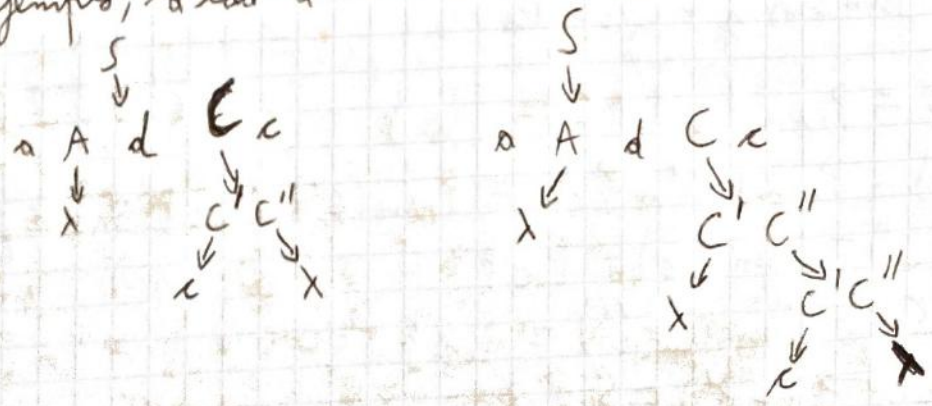
A reemplaza a $(c?\,aB \mid aB)^*$ y $C\,a\,(c?)^+$
A' = $c?\,aB \mid aB$
C' = $c?$
C'' = $(c?)^*$

B' reemplaza $(b,)^*$

Esta gramática veremos que es ambigüa:

Usaré la proposición vista en clase que dice que
$G = \langle V_N, V_T, P, S \rangle$ es ambigua si y sólo si
$\exists\ A \in V_N,\ \alpha, \beta \in V_T^*$ tal que $S \Rightarrow^* \alpha A \beta$
y $\exists\ \gamma \in$

Por ejemplo, la cadena a d c c tiene al menos 2 derivaciones



Como es ambigua, G no es ELL(1). Además, tiene otros problemas que arreglaré en G'.

Sea $G' = \langle \{S, B\}, \{a, b, c, d\}, S, P' \rangle$ con P':

$$S \to a\,(c?\,aB)^*\,d\,c^+$$
$$B \to b\,(,\,b)^*$$

Notas que $(c?\,aB \mid aB)$ y $(c?\,aB)$ generan el mismo lenguaje, y lo mismo entre $(c?)^+\,c$ y c^+ ($\lambda \in L((c?)^+$, por lo que ambas generan cadenas de 1 o más c) y entre $(b,)^*\,b$ y $b\,(,\,b)^*$.

Para ver que sea ELL(1), veo la gramática des-extendida:

$S \to a\,A\,d\,C$ — A reemplaza $(c?\,aB)^*$ y C a c^+

$A \to C'\,aBA \mid \lambda$ — C' " $c?$

$C' \to c \mid \lambda$

$C \to c\,C''$ — C'' reemplaza a c^*

$B \to b\,B'$

$B' \to ,\,b\,B' \mid \lambda$

$C'' \to c\,C'' \mid \lambda$

Calculo Primeros y Siguientes

$Prim(S) = Prim(a\,A\,d\,C) = \{a\}$

$Prim(A) = Prim(C'\,aBA) = Prim(C') \cup \{a\} = \{c, a\}$ — C' es anulable

$Prim(C') = Prim(c) = \{c\}$

$Prim(C) = Prim(c\,C'') = \{c\}$

$Prim(B) \to Prim(b\,B') = \{b\}$

$Prim(B') = Prim(, b\ B') = \{,\}$
$Prim(C'') = Prim(c\ C'') = \{c\}$

Para calcular Siguientes, arranco con $Sig(S) = \{\$\}$

Por $S \to a\ A\ d\ C$, $Sig(A) = \{d\}$
$Sig(C) = Sig(S) = \{\$\}$

Por $A \to C'\ a\ B\ A$, $Sig(C') = \{a\}$ → Anulable
$Sig(B) = Prim(A) \cup Sig(A) = \{c, a\} \cup \{d\}$
$= \{a, c, d\}$

Por $C \to c\ C''$, $Sig(C'') = Sig(C) = \{\$\}$
Por $B \to b\ B'$, $Sig(B') = Sig(B) = \{a, c, d\}$
Una segunda pasada del algoritmo no cambia Siguientes.

Entonces, calculo SD:

$SD(S \to a\ A\ d\ C) = Prim(a\ A\ d\ C) = \{a\}$
$SD(A \to C'\ a\ B\ A) = Prim(C'\ a\ B\ A) = \{c, a\}$
$SD(A \to \lambda) = Sig(A) = \{d\}$
$SD(C' \to C) = \{c\}$
$SD(C' \to \lambda) = Sig(C') = \{a\}$
$SD(C \to c\ C'') = \{c\}$
$SD(B \to b\ B') = \{b\}$
$SD(B' \to , b\ B') = \{,\}$
$SD(B' \to \lambda) = Sig(B') = \{a, c, d\}$
$SD(C'' \to c\ C'') = \{c\}$

$$SP(C'' \to \lambda) = Sig(C'') = \{\$\}$$

Luego, como no hay conflictos, G' es $ELL(1)$.

↓ viendo las intersecciones de SP con mismo símbolo en la cabeza de la producción.

b)

```
S() {
    match("a")
    while (tc in {"c","a"}) {
        if (tc in {"c"}) {
            match("c")
        }
        match("a")
        B()
    }
    match("d")
    do {
        match("c")
    } while (tc in {"c"});
    accept
}
```

```
B() {
    match("b")
    while(tc in {",", "}) {
        match(",")
        match("b")
    }
}
```